

## 7 GRAPHICS

OPL graphics allows you, for example, to:

- Draw lines and boxes.
- Fill areas with patterns.
- Display text in a variety of styles, at any position on the screen.
- Scroll areas of the screen.
- Manipulate windows and bit patterns.
- Read data back from the screen.

On the Series 5 you can additionally,

- Draw circles and ellipses.
- Set the pen width.

You can draw using black, grey and white.

Graphics keywords begin a **G**. In this manual a lower case **g** is used - for example, **gBOX** - but you can type them using upper or lower case letters.



Some graphics keywords are mentioned only briefly in this chapter. For more details about them, see the 'Alphabetic Listing' chapter.

## Simple graphics

The dimensions of the Psion screens are as follows:

| Series 5  | Series 3c | Siena     |
|-----------|-----------|-----------|
| 640 × 240 | 480 × 160 | 240 × 160 |

These points are sometimes referred to as *pixels*.

Each pixel is identified by two numbers, giving its position across and down from the top left corner of the screen. 0,0 denotes the pixel in the top left corner; 2,1 is the pixel 2 points across and one down, and so on. 639,239 is the pixel in the bottom right corner on the Series 5, for example.

Note that these co-ordinates are very different to the cursor positions set by the AT command.

OPL maintains a *current position* on the screen. Graphics commands which draw on the screen generally take effect at this position. Initially, the current position is the top left corner, 0,0.

Î On the Series 5, colour modes allowing the use of 2, 4 or 16 colours are available. These colours are mapped to grey shades on a non-colour screen.

Ì On the Series 3c and Siena, you can draw using black, grey and white although grey is not accessible by default (it has to be switched on explicitly). See the 'Drawing in grey' section below for further details.

### DRAWING LINES

Here is a simple procedure to draw a horizontal line in the middle of the screen:

```
PROC lines:
    gMOVE 180,80
    gLINEBY 120,0
    GET
ENDP
```

gMOVE moves the current position by the specified amount. In this case, it moves the current position 180 pixels right and 80 down, from 0,0 to 180,80. It does not display anything on the screen.

gLINEBY (g-line-by) draws a line from the current position (just set to 180,80) to a point at the distance you specify - in this case 120 to the right and 0 down, i.e. 300,80.

Î The Series 5 never draws the end point of lines: for gLINEBY dx%, dy%, point gX+dx%, gY+dy% is not drawn. Note, however, that OPL specially plots the point when the start and end-point coincide.

Ì When drawing a horizontal line, as in the above example, the line that is drawn **includes** the pixel with the lower x co-ordinate and **excludes** the pixel with the higher x co-ordinate. Similarly, when drawing a vertical line, the line **includes** the pixel with the lower y co-ordinate and **excludes** the pixel with the higher y co-ordinate.

When drawing a diagonal line, the co-ordinates of the end pixels are turned into a rectangle. The top left pixel lies inside the boundary of this rectangle and the bottom right pixel lies outside it. The line drawing algorithm then fills in those pixels that are intersected by a mathematical line between the corners of the rectangle. Thus the line will be drawn minus one or both end points.

gLINEBY also has the effect of moving the current position to the end of the line it draws.

With both gMOVE and gLINEBY, you specify positions **relative to the current position**. Most OPL graphics commands do likewise. gMOVE and gLINEBY, however, do have corresponding commands which use absolute pixel positions. gAT moves to the pixel position you specify; gLINETO draws a

line from the current position to an absolute position. The horizontal line procedure could instead be written:

```
PROC lines:
    gAT 180,80
    gLINETO 300,80
    GET
ENDP
```

gAT and gLINETO may be useful in very short graphics programs, and gAT is always the obvious command for moving to a particular point on the screen, before you start drawing. But once you do start drawing, use gMOVE and gLINEBY. They make it much easier to develop and change programs, and allow you to make useful graphics procedures which can display things anywhere you set the current position. Almost all graphics drawing commands use relative positioning for these reasons.

### Drawing dots

You can set the pixel at the current position with gLINEBY 0,0.

### Right and down, left and up

gMOVE and gLINEBY find the position to use by adding the numbers you specify on to the current position. If the numbers are positive, it moves to the right and down the screen. If you use negative numbers, however, you can specify positions to the left of and/or above the current position. For example, this procedure draws the same horizontal line as before, then another one above it:

```
PROC lines2:
    gMOVE 180,80
    gLINEBY 120,0
    gMOVE 0,-20
    gLINEBY -120,0
    GET
ENDP
```

The first two program lines are the same as before. gLINEBY moves the current position to the end of the line it draws, so after the first gLINEBY the current position is 300,80. The second gMOVE moves the current position **up** by 20 pixels; the second gLINEBY draws a line to a point 120 pixels to the **left**.

Î The end pixel is never set;

Ì For horizontal and vertical lines, the right-hand/bottom pixel is not set. For diagonal lines, the right-most and bottom-most pixels are not set; these may be the same pixel.

### Going off the screen

No error is reported if you try to draw off the edge of the screen. It is quite possible to leave the current position off the screen - for example, gLINETO 650,80 will draw a line from the current position to some point on the right-hand screen edge, but the current position will finish as 650,80.

There's no harm in the current position being off the screen. It allows you to write procedures to display a certain pattern at the current position, and not have to worry whether that position is too close to the screen edge for the whole pattern to fit on.

### Clearing the screen

gCLS clears the screen.

## DRAWING IN GREYS ON THE SERIES 5

### Colour modes

On the Series 5, OPL supports various *colour modes*:

- 2-colour mode (black and white). This is stored as 1 bit per pixel (bpp).
- 4-colour mode (white, light grey, dark grey and black). This is stored as 2 bpp.
- 16-colour mode (white, 14 greys and black). This is stored as 4 bpp.

By default the screen is in 4-colour mode, so black, white and two greys are automatically available. To enable drawing in 16 colours, you need to use `DEFAULTWIN 2` at the start of your program. (Note that this also clears the screen.) 16-colour mode is not automatically available because the hardware switches to 16-colour mode if any window displayed has this mode and 16-colour mode uses twice the memory and much more power than 4-colour mode. So the default ensures that programs that do not need to use 16 colours are not unnecessarily penalised.

The display power consumption is dependent on both the colour mode and the pattern on the display, as follows:

- Colour mode: power consumption doubles from 1 bpp to 2 bpp and again from 2 bpp to 4 bpp.
- Pattern: the worst power consumption for the display is produced by a checker board pattern.

Grey areas also increase the power consumption considerably.

Overall the current taken by the display is between 25% and 60% of the total idle current:

- 25%: 2 bpp plain screen (e.g. plain Word screen).
- 40%: 2 bpp grey areas on screen (e.g. Calc screen).
- 60%: 4 bpp bitmap on the screen.

It is difficult to be more precise since the power consumption is very dependent on what is being displayed.

To set the colour used for graphics, use `gCOLOR red%,green%,blue%`. The `red%`, `green%` and `blue%` values specify a colour, which will be mapped to white, black or one of the greys on non-colour screens. Note that if the values of `red%`, `green%` and `blue%` are equal, then a pure grey results in a 16-colour window, ranging from black (0) to white (255).

Note that if you use `gCOLOR` in 4-colour mode, colours with shades between the four colours available will appear *dithered*, that is areas of the colour will have some pixels set to one colour and some to another so as to give the appearance of a colour between the two colours used. If `gCOLOR` is used in 2-colour mode, light greys will be mapped to white and dark greys to black.

`DEFAULTWIN 1` disables the use of 16 colours again, returning to 4-colour mode and also clearing the screen. If you do not wish to use any greys then you should use `DEFAULTWIN 0` to use 2-colour mode. N.B. This is in fact implemented by mapping dark grey to black and light grey to white.

`DEFAULTWIN` does not effect `PRINT` statements - it applies only to graphics and graphics text (see `gPRINT` later).

Constants for the modes of `DEFAULTWIN` are supplied in `Const.opb`. See the 'Calling Procedures' chapter for details of how to use this file and see Appendix E for a listing of it.

### No concept of a grey plane: grey is just one of the colours

There is no concept of a grey plane on the Series 5 (see the Series 3c description below): `gGREY mode%` just draws in grey (unless `mode%=0`, when it draws in black).

## DRAWING IN GREY ON THE SERIES 3c AND SIENA

### Initialising for the use of grey

To draw in grey you need to use `DEFAULTWIN 1` at the start of your program. (Note that this clears the screen.) Grey is not automatically available because it requires twice the memory (and takes longer to scroll or move) compared to having just black. So programs that do not need to use grey are not unnecessarily penalised.

`DEFAULTWIN 0` disables the use of grey again, also clearing the screen.



It is not possible to have a screen using grey only.

`DEFAULTWIN 1` does not cause `PRINT` to print in grey - it applies only to graphics and graphics text (see `gPRINT` later).

When you use `DEFAULTWIN 1` the existing black-only screen is cleared and replaced by one which contains a *black plane* **and also** a *grey plane*. The black plane is also sometimes called the normal plane. These are referred to as 'planes' because intuitively it is simplest to think of there being a plane of black pixels **in front of** (or on top of) a plane of grey pixels, with any grey only ever visible if the black pixel in front of it is clear.

**If you draw a pixel using both black and grey, it will appear black. If you then clear the black plane only, the same pixel will appear grey.** If you draw a pixel using grey only it will appear grey unless it is already black, in which case it is effectively hidden behind the black plane.

If you need to use grey, you are recommended to use `DEFAULTWIN 1` once and for all at the start of your program. One reason is because `DEFAULTWIN` can fail with a 'No system memory' error and it is unlikely that you would want to continue without grey after trying to enable it.



Note that `gXBORDER`, `gBUTTON` and `gDRAWOBJECT` all use grey and therefore can only be used when grey is enabled. If grey is not enabled, they raise a 'General failure' error.

### Using grey

Once you have used `DEFAULTWIN 1` you can use the `gGREY` command to set which plane should be used for all subsequent graphics drawing (until the next use of `gGREY`).

`gGREY 0` draws to the black plane only.

`gGREY 1` draws to the grey plane only.

`gGREY 2` draws to both planes.

`gGREY 1` and `gGREY 2` raise an error if the current window does not have a grey plane.

As mentioned earlier, when you set a pixel using both black and grey, the pixel appears black because the black plane is effectively in front of the grey plane. So drawing to both planes is generally only used for clearing pixels. For example, if your screen has both black and grey pixels, `gCLS` will clear the pixels only in the plane selected by `gGREY`. To clear the whole screen with `gCLS`, you therefore need `gGREY 2`.

To draw in grey when the pixels to which you are drawing are currently black, you first need to clear the black.

A pixel will appear white only if it is clear in both planes.

### Example

The following procedure initialises the screen to allow grey, draws a horizontal line in grey, another below it in black only and a third below it in both black and grey. Pressing a key clears the black plane only, revealing the grey behind the black in the bottom line and clearing the middle line altogether.

```
PROC exgrey:
```

```
    DEFAULTWIN 1
```

```
    REM enable grey
```

```

gAT 0,40 :gGREY 1 :gLINEBY 480,0      REM grey only
gAT 0,41 :gLINEBY 480,0
gAT 0,80 :gGREY 0 :gLINEBY 480,0      REM black only
gAT 0,81 :gLINEBY 480,0
gAT 0,120 :gGREY 2 :gLINEBY 480,0     REM both planes
gAT 0,121 :gLINEBY 480,0
GET
gGREY 0                                REM black only
gCLS                                    REM clear it
GET
ENDP

```

## OVERWRITING PIXELS

### Drawing rectangles

The gBOX command draws a box outline. For example, gBOX 100,20 draws a box from the current position to a point 100 pixels to the right and 20 down. If the current position were 200,40, the four corners of this box would be at 200,40, 300,40, 300,60 and 200,60.

Î If you have used gCOLOR as described earlier, the box is drawn in the colour selected.

Ì If you have used DEFAULTTWIN 1 and gGREY as described earlier, the box is drawn to the black and/or grey plane as selected.

gBOX does not change the current position.

gFILL draws a filled box in the same way as gBOX draws a box outline, but it has a third argument to say which pixels to set. If set to 0, the pixels which make up the box would be set. If set to 1, pixels are cleared; if set to 2, they are inverted, that is, pixels already set on the screen become cleared, and vice versa. The values 1 and 2 are used when **overwriting** areas of the screen which already have pixels set.

Ì If you have used DEFAULTTWIN 1 and gGREY as described earlier, the filled box will be set, cleared or inverted in the black and/or grey plane as selected. Once again, it helps to think of the pixels being set or clear in each plane independently: so clearing the pixel in the black plane reveals the grey plane behind it where the pixel may be set or clear.

So with gGREY 1 set for drawing to the grey plane only, inverting the pixels in the filled box will change the grey plane only - black pixels are left alone but clear or grey pixels are inverted to grey and clear pixels respectively. Similarly, inverting the black plane changes clear pixels to black, but “clearing” black pixels displays grey if the pixel is set in the grey plane.

This procedure displays a “robot” face, using gFILL to draw set and cleared boxes:

```

PROC face:
gFILL 120,120,0                        REM set the entire face
gMOVE 10,20 :gFILL 30,20,1            REM left eye
gMOVE 70,0 :gFILL 30,20,1            REM right eye
gMOVE -30,30 :gFILL 20,30,1          REM nose
gMOVE -20,40 :gFILL 60,20,1          REM mouth
GET
ENDP

```

Before calling such a procedure, you would set the current position to be where you wanted the top left corner of the head.

You could make the robot wink with the following procedure, which inverts part of one eye:

```
PROC wink:
    gMOVE 10,20                REM move to left eye
    gFILL 30,14,2              REM invert most of the eye
    PAUSE 10
    gFILL 30,14,2              REM invert it back again
    GET
ENDP
```

Again, you would set the current position before calling this.

The gPATT command can be used to draw a shaded filled rectangle. To do this, use -1 as its first argument, then the same three arguments as for gFILL - width, height, and overwrite method. Overwrite methods 0, 1 and 2 apply only to the pixels which are 'on' in the shading pattern. Whatever was on the screen may still show through, as those pixels which are 'clear' in the shading pattern are left as they were.

To completely overwrite what was on the screen with the shaded pattern, gPATT has an extra overwrite method of 3. So, for example, gPATT -1,120,120,3 in the first procedure would have displayed a shaded robot head, whatever may have been on the screen.

Ì Again, the shaded pattern will be drawn in grey if you have selected the grey plane only using gGREY 1. And again, if you are writing to the black plane only, any pixels set in the grey plane can be seen if the corresponding pixels in the black plane are clear.

### Overwriting with any drawing command

By using the gGMODE command, any drawing command such as gLINEBY or gBOX can be made to clear or invert pixels, instead of setting them. gGMODE determines the effect of **all** subsequent drawing commands.

The values are the same as for gFILL: gGMODE 1 for clearing pixels, gGMODE 2 for inverting pixels, and gGMODE 0 for setting pixels again. (0 is the initial setting.)

For example, some white lines can give the robot a furrowed brow:

```
PROC brow:
    gGMODE 1                    REM gLINEBY will now clear pixels
    gMOVE 10,8 :gLINEBY 100,0
    gMOVE 0,4 :gLINEBY -100,0
    gGMODE 0
    GET
ENDP
```

Ì The setting for gGMODE applies to the planes selected by gGREY. With gGREY 1 for instance, gGMODE 1 would cause gLINEBY to clear pixels in the grey plane and gGMODE 0 to set pixels in the grey plane.

Î Constants for the modes are supplied in Const.oph. See the 'Calling Procedures' chapter for details of how to use this file and see Appendix E for a listing of it.

## Other drawing keywords

For more details of these keywords, see the ‘Alphabetic Listing’ chapter.

- **gBUTTON**: draw a 3-D button (a picture of a key, not of an application button) enclosing supplied text. The button can be raised, depressed or sunken. On the Series 5, the button may also enclose a *bitmap* with a *mask* (see the ‘Bitmaps’ section below).
- **gBORDER**, **gXBORDER**: draw 2-D/3-D borders.
- **gINVERT**: invert a rectangular area, except for its four corner pixels.
- **gCOPY**: copy a rectangular area from one position on the screen to another. On the Series 3c, both black and grey planes are copied.
- **gSCROLL**: move a rectangular area from one position on the screen to another, or scroll the contents of the screen in any direction. On the Series 3c, both black and grey planes are moved.
- **gPOLY**: draw a sequence of lines.
- **gCIRCLE**, **gELLIPSE**: draw a circle or ellipse which can be filled or empty.
- **gSETPENWIDTH**: draw with a different pen width.
- **gDRAWOBJECT**: draw a *graphics object*. This can be used to draw the “lozenge” used to display the words ‘City’ and ‘Home’ in the World application.

Note that commands such as **gSCROLL**, which move existing pixels, affect both black and grey planes. **gGREY** only restricts drawing and clearing of pixels.

## GRAPHICAL TEXT

### Displaying text with gPRINT

The **PRINT** command displays text in one font, in a screen area controlled by the **FONT** or **SCREEN** commands. You can, however, display text in a variety of fonts and styles, at any pixel position, with **gPRINT**.

**gPRINT** also allows you to draw text in grey if you have used the **gCOLOR** command previously.

**gPRINT** also lets you draw text to the grey plane, if you have used **DEFAULTWIN** and **gGREY** (discussed earlier).



You can (to a lesser degree on the Series 3c) control the font and style used by OPL’s other text-drawing keywords, such as **PRINT** and **EDIT**. See ‘The text and graphics windows’ section at the end of this chapter.

**gPRINT** is a graphical version of **PRINT**, and displays a list of expressions in a similar way. Some examples:

```
gPRINT "Hello",name$
```

```
gPRINT a$
```

```
gPRINT "Sin(PI/3) is",sin(pi/3)
```

Unlike **PRINT**, **gPRINT** does not end by moving to a new line. A comma between expressions is still displayed as a space, but a semicolon has no effect. **gPRINT** used on its own does nothing.

The first character displayed has its left side and baseline at the current position. The baseline is like a line on lined note paper graphically, this is the horizontal line which includes the lowest pixels of upper case characters. Some characters, such as ‘g’, ‘j’, ‘p’, ‘q’ and ‘y’, set pixels below the baseline.



After using gPRINT, the current position is at the end of the text so that you can print something else immediately beyond it. As with other graphics keywords, no error is reported if you try to display text off the edge of the screen.

While `CURSOR ON` displays a flashing cursor for ordinary text displayed with `PRINT`, `CURSOR 1` switches on a cursor for graphical text which is displayed at the current position. `CURSOR OFF` removes either cursor.

## Fonts

The gFONT command sets the font to be used by subsequent gPRINT commands.

A large set of fonts which can be used with gFONT is provided in the Psion's ROM. In the following list, Swiss and Arial fonts refer to fonts without serifs while Roman and Times fonts either have serifs (e.g. font 6) or are in a style designed for serifs but are too small to show them (e.g. font 5 on the Series 3c). Courier is a *mono-spaced* font. *Mono-spaced* fonts have characters which all have the same width (and have their 'pixel size' listed as width × height); in *proportional* fonts each character can have a different width.



Fonts 1,2 and 3 are the Series 3 fonts, used when running in compatibility mode. Therefore these fonts are not supported on the Series 5.

| Font number | Series 5 font name | height in pixels | Series 3c font name | size in pixels |
|-------------|--------------------|------------------|---------------------|----------------|
| 1           | -                  | -                | Series 3 normal     | 8              |
| 2           | -                  | -                | Series 3 bold       | 8              |
| 3           | -                  | -                | Series 3 digits     | 6x6            |
| 4           | Courier            | 8                | Mono                | 8x8            |
| 5           | Times              | 8                | Roman               | 8              |
| 6           | Times              | 11               | Roman               | 11             |
| 7           | Times              | 13               | Roman               | 13             |
| 8           | Times              | 15               | Roman               | 16             |
| 9           | Arial              | 8                | Swiss               | 8              |
| 10          | Arial              | 11               | Swiss               | 11             |
| 11          | Arial              | 13               | Swiss               | 13             |
| 12          | Arial              | 15               | Swiss               | 16             |
| 13          | Tiny (mono)        | 4                | Mono                | 6x6            |

The special font number &9a (\$9a on the Series 3c) is set aside to give a machine's default graphics font; this is the font used initially for graphics text. The default font is 12 (Arial 15) for the Series 5 and 11 (Swiss 13) for the Series 3c. So `gFONT 12 (11)` or `gFONT &9a ($9a on the Series 3c)` both set the standard font, which gPRINT normally uses.



On the Series 5, fonts are identified by a 32-bit UID, rather than a 16-bit value representing the font position in the ROM as on the Series 3c. Series 5 OPL does however provide a mapping where possible between Series 3c OPL font IDs and Series 5 OPL font UIDs, but there are inevitably some incompatibilities.

Therefore gFONT takes a long integer argument rather than an integer. As well as being able to use the font IDs 4 to 13 (which are automatically converted to long integers for all keywords taking font IDs), you can also directly specify the fonts by UID by using the definitions listed in the header file Const.opb. (See the 'Calling Procedures' chapter for details of how to use this file and see Appendix E for a listing of it.) A wider range of fonts is also available, as you will see from Const.opb. These are Arial and Times proportional fonts and Courier mono-spaced font,

each with sizes 8, 11, 13, 15, 18, 22, 27 and 32, Squashed font (which are used for the toolbar in bold style) and Tiny fonts.

For example, normal Arial proportional font with height 8 pixels has UID given by,

```
CONST KFontArialNormal8&=268435954
```

so you could use,

```
INCLUDE "Const.oph"
```

```
...
```

```
gFONT KFontArialNormal8&
```

See gINFO32 (for the Series 5) or gINFO (for the Series 3c) in the 'Alphabetic Listing' chapter if you need to find out more information about fonts.

The following programs shows you examples of the fonts. (!!! is displayed to emphasise the mono-spaced fonts):

Î This program displays a variety of fonts, using both the OPL codes for them and their UIDS (the constants are defined in Const.oph - see above). The two screens should be identical.

```
INCLUDE "Const.oph"
```

```
PROC fonts:
```

```
showfont:(4,15,"Courier 8")
```

```
showfont:(5,25,"Times 8")
```

```
showfont:(6,38,"Times 11")
```

```
showfont:(7,53,"Times 13")
```

```
showfont:(8,71,"Times 15")
```

```
showfont:(9,81,"Arial 8")
```

```
showfont:(10,94,"Arial 11")
```

```
showfont:(11,109,"Arial 13")
```

```
showfont:(12,127,"Arial 15")
```

```
showfont:(13,135,"Tiny 44")
```

```
GET :GCLS
```

```
showfontbyuid:(KFontCourierNormal8&,15,"Courier 8")
```

```
showfontbyuid:(KFontTimesNormal8&,25,"Times 8")
```

```
showfontbyuid:(KFontTimesNormal11&,38,"Times 11")
```

```
showfontbyuid:(KFontTimesNormal13&,53,"Times 13")
```

```
showfontbyuid:(KFontTimesNormal15&,71,"Times 15")
```

```
showfontbyuid:(KFontArialNormal8&,81,"Arial 8")
```

```
showfontbyuid:(KFontArialNormal11&,94,"Arial 11")
```

```
showfontbyuid:(KFontArialNormal13&,109,"Arial 13")
```

```
showfontbyuid:(KFontArialNormal15&,127,"Arial 15")
```

```
showfontbyuid:(KFontTiny4&,135,"Tiny 4")
```

```
ENDP
```

```

PROC showfont:(font%,y%,str$)
    gFONT font%
    gAT 20,y% :gPRINT font%
    gAT 50,y% :gPRINT str$
    gAT 150,y% :gPRINT "!!!"
ENDP

PROC showfontbyuid:(font&,y%,str$)
    gFONT font&
    gAT 20,y% :gPRINT font%
    gAT 50,y% :gPRINT str$
    gAT 150,y% :gPRINT "!!!"
ENDP

```

Ì PROC fonts:

```

    showfont:(4,15,"Mono 8x8")
    showfont:(5,25,"Roman 8")
    showfont:(6,38,"Roman 11")
    showfont:(7,53,"Roman 13")
    showfont:(8,71,"Roman 16")
    showfont:(9,81,"Swiss 8")
    showfont:(10,94,"Swiss 11")
    showfont:(11,109,"Swiss 13")
    showfont:(12,127,"Swiss 16")
    showfont:(13,135,"Mono 6x6")
    GET
ENDP

```

```

PROC showfont:(font%,y%,str$)
    gFONT font%
    gAT 20,y% :gPRINT font%
    gAT 50,y% :gPRINT str$
    gAT 150,y% :gPRINT "!!!"
ENDP

```

### Text style

The gSTYLE command sets the text style to be used by subsequent gPRINT commands.

Choose from these styles:

```

gSTYLE 1      bold
gSTYLE 2      underlined
gSTYLE 4      inverse
gSTYLE 8      double height
gSTYLE 16     mono
gSTYLE 32     italic

```

Î Constants for the styles are supplied in Const.opb. See the ‘Calling Procedures’ chapter for details of how to use this file and see Appendix E for a listing of it.

The ‘mono’ style is not proportionally spaced - each character is displayed with the same width, in the same way that PRINT displays characters (by default). A proportional font can be displayed as a mono-spaced font by setting the ‘mono’ style. See the previous section for the list of mono-spaced and proportional fonts.

 It is inefficient to use the ‘mono’ style to display a font which is already mono-spaced.

You can combine these styles by adding the relevant numbers together. gSTYLE 12 sets the text style to inverse and double-height (4+8=12). Here’s an example of this style:

```

PROC style:
    gAT 20,50 :gFONT 11
    gSTYLE 12 :gPRINT "Attention!"
    GET
ENDP

```

Use gSTYLE 0 to reset to normal style.

The bold style provides a way to make any font appear bold. Except for the smaller fonts on the Series 3c, most Psion fonts look reasonably bold already. Note that using the bold style sometimes causes a change of font; if you use gINFO you may see the font name change.

Î Note that fonts which are always bold are available on the Series 5. Using a bold style with these fonts results in a double bold font. It is not necessary to use these fonts to produce bold font: you can of course use just the normal fonts in a bold style.

## Overwriting with gPRINT

gPRINT normally displays text as if writing it with a pen - the pixels that make up each letter are set, and that is all. If you’re using areas of the screen which already have some pixels set, or even have all the pixels set, use gTMODE to change the way gPRINT displays the text.

gTMODE controls the display of text in the same way as gGMODE controls the display of lines and boxes. The values you use with gTMODE are similar to those for gGMODE: gTMODE 1 for clearing pixels, gTMODE 2 for inverting pixels, and gTMODE 0 for setting pixels again. There is also gTMODE 3 which sets the pixels of each character while clearing the character’s background. This is very useful as it guarantees that the text is readable.

Î As for gGMODE, the setting for gTMODE applies to the planes selected by gGREY. With gGREY 1 for instance, gTMODE 1 would cause gPRINT to clear pixels in the grey plane and gTMODE 0 to set pixels in the grey plane.

Î Constants for the modes of gTMODE are supplied in Const.opb. See the ‘Calling Procedures’ chapter for details of how to use this file and see Appendix E for a listing of it.

These procedures (for the Series 5 and the Series 3c respectively) shows the various effects possible via gTMODE:

```

Î PROC tmode:
    DEFAULTTWIN 2
    gFONT 11 :gSTYLE 0
    gAT 160,0 :gFILL 160,80,0          REM Black box
    gAT 220,0 :gFILL 40,80,1          REM White box
    gAT 180,20 :gTMODE 0 :gPRINT "ABCDEFGHIIJK"
    gAT 180,35 :gTMODE 1 :gPRINT "ABCDEFGHIIJK"
    gAT 180,50 :gTMODE 2 :gPRINT "ABCDEFGHIIJK"
    gAT 180,65 :gTMODE 3 :gPRINT "ABCDEFGHIIJK"
    gCOLOR $50,$50,$50
    gAT 160,80 :gFILL 160,80,0          REM Grey box
    gAT 220,80 :gFILL 40,80,1          REM White box
    gAT 180,100 :gTMODE 0 :gPRINT "ABCDEFGHIIJK"
    gAT 180,115 :gTMODE 1 :gPRINT "ABCDEFGHIIJK"
    gAT 180,130 :gTMODE 2 :gPRINT "ABCDEFGHIIJK"
    gAT 180,145 :gTMODE 3 :gPRINT "ABCDEFGHIIJK"
    GET
ENDP

```

```

Î PROC tmode:
    DEFAULTTWIN 1          REM enable grey
    gFONT 11 :gSTYLE 0
    gAT 160,0 :gFILL 160,80,0          REM Black box
    gAT 220,0 :gFILL 40,80,1          REM White box
    gAT 180,20 :gTMODE 0 :gPRINT "ABCDEFGHIIJK"
    gAT 180,35 :gTMODE 1 :gPRINT "ABCDEFGHIIJK"
    gAT 180,50 :gTMODE 2 :gPRINT "ABCDEFGHIIJK"
    gAT 180,65 :gTMODE 3 :gPRINT "ABCDEFGHIIJK"
    gGREY 1
    gAT 160,80 :gFILL 160,80,0          REM Grey box
    gAT 220,80 :gFILL 40,80,1          REM White box
    gAT 180,100 :gTMODE 0 :gPRINT "ABCDEFGHIIJK"
    gAT 180,115 :gTMODE 1 :gPRINT "ABCDEFGHIIJK"
    gAT 180,130 :gTMODE 2 :gPRINT "ABCDEFGHIIJK"
    gAT 180,145 :gTMODE 3 :gPRINT "ABCDEFGHIIJK"
    GET
ENDP

```

### Other graphical text keywords

- gPRINTB: display text left aligned, right aligned or centred, in a cleared box. The gTMODE setting is ignored.

Î With gGREY 1, only grey background pixels in the box are cleared and with gGREY 0, only black pixels; with gGREY 2 all background pixels in the box are cleared.

- gXPRINT: display text underlined/highlighted.
- gPRINTCLIP: display text clipped to whole characters.
- gTWIDTH: find width required by text.

All of these keywords take the current font and style into account, and work on **a single string**. On the Series 5, they display text in the colour specified by gCOLOR. On the Series 3c, they display the text in black or grey according to the current setting of gGREY.

### WINDOWS

So far, you've used the whole of the screen for displaying graphics. You can, however, use *windows* - rectangular areas of the screen.

Î  Sprites (described in the 'Using OPXs' chapter) can display non-rectangular shapes.

OPL allows a program to use up to sixty-four windows at any one time.

Î  Sprites (described in the 'Advanced topics' chapter) can display non-rectangular shapes.

OPL allows a program to use up to eight windows at any one time.

### Window IDs and the default window

Each window has an ID number, allowing you to specify which window you want to work with at any time.

When a program first runs, it has one window called the *default window*. Its ID is 1, it is the full size of the screen, and initially all graphics commands operate on it. (This is why '0,0' has so far referred to the top left of the screen: it is true for the default window.)

Other windows you create will have IDs from 2 to 64 (2 to 8 on the Series 3c). **When you make another window it becomes the current window, and all subsequent graphics commands operate on it.**

The first half of this chapter used only the default window. However, everything actually applies to the current window. For example, if you make a small window current and try to draw a very long line, the current position moves off past the window edge, and only that part of the line which fits in the **window** is displayed.

### Graphics keywords and windows

For OPL graphics keywords, **positions apply to the window you are using at any given time**. The point 0,0 means the top left corner of the current window, not the top left corner of the screen.

Î Each window can be created in any of the 3 colour modes by specifying the last argument in the gCREATE command (see below and the 'Alphabetic Listing' chapter). gCOLOR can be used to specify the current pen colour and gSETPENWIDTH to specify the pen width. The default is 4-colour mode, as for the default window.

For the default window, the special command DEFAULTWIN is required to change colour modes because that window is automatically created for you in 4-colour mode; DEFAULTWIN clears the default window and resets colour mode to that specified. All other windows must be **created** in the colour mode in which they are required: it may not be changed once they are created.

Once a window has been created with a certain colour mode, colours specified by gCOLOR work in the exactly the same way as in the default window.

Î Each window can be created with a grey plane if required, in which case gGREY is used to specify whether the black plane, the grey plane or both should be used for all subsequent graphics commands until the next call to gGREY, exactly as described in the first half of this chapter.

For the default window, the special command DEFAULTWIN is required to enable grey because that window is automatically created for you with only a black plane; DEFAULTWIN 1 closes the default window and creates a new one which has a grey plane. All other windows must be **created** with a grey plane if grey is required.

Once a window has been created with a grey plane, grey is used in precisely the same way as in the default window with grey enabled: gGREY 0 directs all drawing to the black plane only, gGREY 1 to the grey plane only and gGREY 2 to both planes. gGREY 1 and gGREY 2 raise an error if the current window does not have a grey plane.

gGREY, gGMODE, gTMODE, gFONT and gSTYLE can all be used with created windows in exactly the same way as with the default window, as described earlier. **They change the settings for the current window only; all the settings are remembered for each window.**

### Creating new windows

The gCREATE function sets up a new window on the screen. It returns an ID number for the window. Whenever you want to change to this window, use gUSE with this ID.

Î You can create a window with any of the three colour modes by specifying the last optional parameter to gCREATE.

Here is an example using gCREATE and gUSE, contrasting the point 20,20 in the created window with 20,20 in the default window.

PROC windows:

```
LOCAL id%
id%=gCREATE(60,40,320,30,1,2)          REM 16-colour mode
gBORDER 0 :gAT 20,20 :gLINEBY 0,0
gPRINT " 20,20 (new)"
GET
gUSE 1 :gAT 20,20 :gLINEBY 0,0
gPRINT " 20,20 (default)"
GET
gUSE id%
gCOLOR $88,$88,$88                     REM mid grey
gPRINT " Back"
gCOLOR 0,0,0,                          REM black
gPRINT " (with 16 colours)"
GET
```

ENDP

The line id%=gCREATE(60,40,320,30,1,2) creates a window with its top left corner at 60,40 on the screen. The window is set to be 320 pixels wide and 30 pixels deep. (You can use any integer values for these arguments, even if it creates the window partially or even totally off the screen.) The fifth argument to gCREATE specifies whether the window should immediately be visible or not; 0 means invisible, 1 (as here) means visible. The sixth argument specifies the

colour mode; 0 means 2-colour mode, 1 means 4-colour mode and 2 (as here) means 16 colour mode. If the sixth argument is not supplied at all (e.g. `id%=gCREATE(60,40,320,30,1)`) the window will have the default 4-colour mode.

**I** You can create a window with only a black plane or with both a black and a grey plane. You cannot create a window with just a grey plane.

Here is an example using `gCREATE` and `gUSE`, contrasting the point 20,20 in the created window with 20,20 in the default window.

PROC windows:

```
LOCAL id%
id%=gCREATE(60,40,240,30,1,1)
gBORDER 0 :gAT 20,20 :gLINEBY 0,0
gPRINT " 20,20 (new)"
GET
gUSE 1 :gAT 20,20 :gLINEBY 0,0
gPRINT " 20,20 (default)"
GET
gUSE id%
gGREY 1                                REM draw grey
gPRINT " Back"
gGREY 0
gPRINT " (with grey)"
GET
ENDP
```

The line `id%=gCREATE(60,40,240,30,1,1)` creates a window with its top left corner at 60,40 on the screen. The window is set to be 240 pixels wide and 30 pixels deep. (You can use any integer values for these arguments, even if it creates the window partially or even totally off the screen.) The fifth argument to `gCREATE` specifies whether the window should immediately be visible or not; 0 means invisible, 1 (as here) means visible. The sixth argument specifies whether the window should have a grey plane or not; 0 means black only, 1 (as here) means black and grey. If the sixth argument is not supplied at all (e.g. `id%=gCREATE(60,40,240,30,1)`) the window will not have a grey plane.

`gCREATE` automatically makes the created window the current window, and sets the current position in it to 0,0. It returns an ID number for this window, which in this example is saved in the variable `id%`.

The `gBORDER 0` command draws a border one pixel wide around the current window. Here this helps show the position and size of the window. (`gBORDER` can draw a variety of borders. You can even display the 3-D style borders seen in menus and dialogs, with the `gXBORDER` keyword.)

The program then sets the pixel at 20,20 in this new window, using `gLINEBY 0,0`.

`gUSE 1` goes back to using the default window. The program then shows 20,20 in this window.

Finally, `gUSE id%` goes back to the created window again, and a final message is displayed, in grey and black.

Note that **each window has its own current position**. The current position in the created window is remembered while the program goes back to the default window. **All the other settings, such as the font, style and grey setting are also remembered.**



## Closing windows

When you've finished with a particular window, close it with `gCLOSE` followed by its ID - for example, `gCLOSE 2`. You can create and close as many windows as you like, as long as there are only 64 (8 on the Series 3c) or fewer open at any one time.

If you close the current window, the default window (ID=1) becomes current.

An error is raised if you try to close the default window.

## When windows overlap

Windows can overlap on the screen, or even hide each other entirely. Use the `gORDER` command to control the foreground/background positions of overlapping windows.

`gORDER 3, 1` sets the window whose ID is 3 to be in the foreground. This guarantees that it will be wholly visible. `gORDER 3, 2` makes it second in the list; unless the foreground window overlaps it, it too will be visible.

Any position greater than the number of windows you have is interpreted as the end of the list.

`gORDER 3, 9` will therefore always force the window whose ID is 3 to the background, behind all others.



**Note in particular that making a window the current window with `gUSE` does not bring it to the foreground.** You can make a background window current and draw all kinds of things to it, but nothing will happen on the screen until you bring it to the foreground with `gORDER`.

When a window is first created with `gCREATE` it always becomes the foreground window as well as the current window.

## Hiding windows

If you are going to use several drawing commands on a particular window, you may like to make it invisible while doing so. When you then make it visible again, having completed the drawing commands, the whole pattern appears on the screen in one go, instead of being built up piece by piece.

Use `gVISIBLE ON` and `gVISIBLE OFF` to perform this function on the current window. You can also make new windows invisible as you create them, by using 0 as the fifth argument to the `gCREATE` command, and you can hide windows behind other windows.

## The graphics cursor in windows

To make the graphics cursor appear in a particular window, use the `CURSOR` command with the ID of the window. It will appear flashing at the current position in that window, provided it is not obscured by some other window.

The window you specify does not have to be the current window, and does not become current; you can have the cursor in one window while displaying graphical text in another. If you want to move to a different window and put the graphics cursor in it, you must use both `gUSE` and `CURSOR`.

Since the default window always has an ID of 1, `CURSOR 1` will, as mentioned earlier, put the graphics cursor in it.

`CURSOR OFF` turns off the cursor, wherever it is.

## Information about your windows

You don't have to keep a complete copy of all the information pertaining to each window you use. These functions return information about the current window:

- `gIDENTITY` returns its ID number.
- `gRANK` returns its foreground/background position, from 1 to 8.
- `gWIDTH` and `gHEIGHT` return its size.
- `gORIGINX` and `gORIGINY` return its screen position.

- $\hat{I}$  gINFO32 returns information about the font, style, colour setting, overwrite modes and cursor in use.
- $\grave{I}$  gINFO returns information about the font, style, grey setting, overwrite modes and cursor in use.
- gX and gY return the current position.

### Other window keywords

- gSETWIN changes the position, and optionally the size, of the current window.
-  You **can** use this command on the default window, if you wish, but you must also use the SCREEN command to ensure that the *text window* the area for PRINT commands to use is wholly contained within the default window. See ‘The text and graphics windows’, later in this chapter.
- gSCROLL scrolls all or part of both black and grey planes of the current window.
- gPATT fills an area in the current window with repetitions of another window, or with a shaded pattern.
- gCOPY copies an area from another window into the current window, or from one position in the current window to another.
-  On the Series 5, it is unadvisable to use gCOPY to copy from windows as it is very slow. It should only be used for copying from bitmaps to windows or other bitmaps.
- gSAVEBIT saves part or all of a window as a *bitmap file*.

- $\grave{I}$  If a window has a grey plane, the planes are saved as two bitmaps to the same file with the black plane saved first and the grey plane saved next. gLOADBIT, described later, can be used to load bitmap files.
- gPEEKLINE reads back a horizontal line of data in the specified mode (for the Series 5: see the ‘Alphabetic Listing’), or from either the black or grey plane (on the Series 3c) of a specified window.

### $\grave{I}$ Copying grey between windows

The commands gCOPY and gPATT can use two windows and therefore special rules are needed for the cases when one window has a grey plane and the other does not.

With gGREY 0 in the destination window, only the black plane of the source is copied.

With gGREY 1 in the destination window, only the grey plane of the source is copied, unless the source has only one plane in which case that plane is used as the source.

With gGREY 2 in the destination window, if the source has both planes, they are copied to the appropriate planes in the destination window (black to black, grey to grey); if the source has only one plane, it is copied to **both** planes of the destination.

## ADVANCED GRAPHICS

This section should provide a taste of some of the more exotic things you can do with OPL graphics.

### Bitmaps

A *bitmap* is an area in memory which acts just like an off-screen window. You can create bitmaps with gCREATEBIT.

- $\hat{I}$  It is possible to create bitmaps in any of the three colour modes by specifying the optional third argument when using gCREATEBIT. The default is 2-colour mode. Note, however, that black and white bitmaps differ from black and white windows. In particular, if you draw in 16 colours to a black and white bitmap then greys appear as dithered black and white, whereas if you draw

exactly the same to a 2-colour graphics window you just get dark greys mapped to black and light greys mapped to white. This enables grey printing on black and white printers.

A further benefit is that the file size will be smaller if the bitmap is saved, with just 1 bpp used for black and white bitmaps.

Ì Note that a bitmap does not have two planes so that gGREY cannot be used.

Bitmaps have the following uses:

- You can manipulate an image in a bitmap before copying it with gPATT or gCOPY to a window on the screen. This is generally faster than manipulating an image in a hidden window.
- You can load *bitmap files* into bitmaps in memory using gLOADBIT, then copy them to on-screen windows using gCOPY or gPATT.

Ì If a black and grey window was saved to file as two bitmaps using gSAVEBIT, you must load them separately into two bitmaps in memory, and copy them one at a time to the respective planes of a window.

**OPL treats a bitmap as the equivalent of a window in most cases:**

- Both are identified by ID numbers. Only one window or bitmap is current at any one time, set by gUSE.
- If you use bitmaps as well as windows, the **total** number must be 64 (8 on the Series 3c) or fewer.
- The top left corner of the current bitmap is still referred to as 0,0, even though it is not on the screen at all.

Together, windows and bitmaps are known as *drawables* - places you can draw to.

Most graphics keywords can be used with bitmaps in the same way as with windows, but remember that a bitmap corresponds to only one plane in a window. Once you have drawn to it, you might copy it to the appropriate plane of a window.

The keywords that can be used with bitmaps include: gLINEBY, gLINETO, gBOX, gFILL, gCIRCLE (Series 5 only), gELLIPSE (Series 5 only), gCOLOR (Series 5 only), gSETPENWIDTH (Series 5 only), gUSE, gBORDER, gCLOSE, gCLS, gCOPY, gMODE, gFONT, gIDENTITY, gPATT, gPEEKLINE, gSAVEBIT, gSCROLL, gTMODE, gWIDTH, gHEIGHT, gINFO32 (Series 5) and gINFO (Series 3). These keywords are described earlier in this chapter.

## Î Masks

There are several keywords that require an understanding of *masks*. In some cases the mask is a bitmap file, e.g. gBUTTON (see earlier in this chapter and also the 'Alphabetic Listing' chapter) and for ICON (see the 'Advanced Topics' chapter), and in some cases it is an integer containing a *bit-mask*, e.g. for POINTERFILTER (see again the 'Advanced Topics' chapter).

In all these cases, however, the principle is the same. The pixels or bits which are set in the mask specify pixels or bits in some other argument which are to be used. Pixels and bits which are clear in the mask specify pixels and bits that are not to be used from the other argument.

For example, when using gBUTTON with identical bitmap and mask, cleared pixels on the bitmap are drawn in the background colour of the button (i.e. they are clear) while set pixels are drawn on the button as they appear on the bitmap. This is generally how buttons on Series 5 toolbars appear.

## Speed improvements

The Psion's screen is usually updated whenever you display anything on it. gUPDATE OFF switches off this feature. The screen will be updated as few times as possible, although you can force an update by using the gUPDATE command on its own. (An update is also forced by GET, KEY and by all graphics keywords which return a value, other than gX, gY, gWIDTH and gHEIGHT).

This can result in a considerable speed improvement in some cases. You might, for example, use `gUPDATE OFF`, then a sequence of graphics commands, followed by `gUPDATE`. You should certainly use `gUPDATE OFF` if you are about to write exclusively to bitmaps.

`gUPDATE ON` returns to normal screen updating.

Î As mentioned previously, a window with both black and grey planes takes longer to move or scroll than a window with only a black plane. So avoid creating windows with unnecessary grey planes.

Also, remember that scrolling and moving windows require every pixel in a window to be redrawn.

The `gPOLY` command draws a sequence of lines, as if by `gLINEBY` and `gMOVE` commands. If you have to draw a lot of lines (or dots, with `gLINEBY 0, 0`), `gPOLY` can greatly reduce the time taken to do so.

### Displaying a running clock

`gCLOCK` displays or removes a running clock showing the system time. The clock can be digital or conventional, and can use many different formats. See the ‘Alphabetic Listing’ chapter for full details.

### User-defined fonts and cursors

If you have a user-defined font you can load it into memory with `gLOADFONT`.

Î `gLOADFONT` returns a file ID, which can be used only with `gUNLOADFONT`. The maximum number of font files which may be loaded at any one time is 16. To use the fonts in a loaded font you need to use the UIDs specified in the font file itself.

Î This returns an ID for the font; use this with `gFONT` to make the font current. The `gUNLOADFONT` command removes a user-defined font from memory when you have finished using it.

You can use four extra arguments with the `CURSOR` command. Three of these specify the ascent, width and height of the cursor. The ascent is the number of pixels (-128 to 127) by which the top of the cursor should be above the baseline of the current font. The height and width arguments should both be between 0 and 255. For example, `CURSOR 1, 12, 4, 14` sets a cursor 4 pixels wide by 14 high in the default window (ID=1), with the cursor top at 12 pixels above the font baseline.

If you do not use these arguments, the cursor is 2 pixels wide, and has the same height and ascent as the current font.

By default the cursor has square corners, is black and is flashing. Supply the fifth argument as 2 for non-flashing or 4 for grey (1 for a rounded cursor is also available on the Series 3c). You can add these together - e.g. use 6 for a grey, non-flashing cursor.

Note that the `gINFO32` and `gINFO` (Series 5 and Series 3c respectively) command returns information about the cursor and font.

### The text and graphics windows

`PRINT` displays mono-spaced text in the *text window*. You can change the text window font (i.e. that used by `PRINT`) using the `FONT` keyword.

Î You can use any of those fonts listed in `Const.oph`; see the ‘Calling Procedures’ for an explanation of how to use this file and Appendix E for a listing of it. Initially Courier 11 is used on the Series 5.

It should be noted that when using the console keywords `PRINT`, `AT`, `SCREEN`, etc. the use of Series 5 proportional fonts such as Arial and Times may produce some unexpected behaviour because it is assumed in all cases that mono-spaced font is being used. The reason for this is so that use of keywords such as `AT`, `SCREEN` in the console is independent of whether the font is proportional or monospaced. An example of this behaviour may be seen when using inverted text:

the rectangle to invert is calculated assuming that the font is mono-spaced, and hence the area inverted is larger than the text printed when using a proportional font. Another example is that a new line will be used before it appears necessary when using a proportional font, since the number of characters which will fit on a line is also calculated assuming the font is mono-spaced.

**I** You can use any of those fonts listed earlier in the chapter in the description of gFONT; initially font 4 is used on the Series 3c.

The text window is in fact part of the default graphics window. If you have other graphics windows in front of the default window, they may therefore hide any text you display with PRINT.

Initially the text window is very slightly smaller than the default graphics window which is full-screen size. They are not the same because **the text window height and width always fits a whole number of characters of the current text window font**. If you use the FONT command to change the font of the text window, this first sets the default graphics window to the maximum size that will fit in the screen (excluding any status window on the Series 3c) and then resizes the text window to be as large as possible inside it.

You can also use the STYLE keyword to set the style for all characters subsequently written to the text window. This allows the mixing of different styles in the text window.

**I** You can only use those styles which do not change the size of the characters - i.e. inverse video and underline. (Any other styles will be ignored.)

Use the same values as listed for gSTYLE, earlier in the chapter.

To find out exactly where the text window is positioned, use SCREENINFO info%(). This sets info%(1)/info%(2) to the number of pixels from the left/top of the default window to the left/top of the text window. (These are called the margins.) info%(7) and info%(8) are the text window's character width and height respectively.



The margins are fully determined by the font being used and therefore change from their initial value only when FONT is used. You cannot choose your own margins. gSETWIN and SCREEN do not change the margins, so you can use FONT to select a font (also clearing the screen), followed by SCREENINFO to find out the size of the margins with that font, and finally gSETWIN and SCREEN to change the sizes and positions of the default window and text window taking the margins into account (see example below). The margins will be the same after calling gSETWIN and SCREEN as they were after FONT.

It is not generally recommended to use both the text and graphics windows. Graphics commands provide much finer control over the screen display than is possible in the text window, so it is not easy to mix the two.

If you do need to use the text window, for example to use keywords like EDIT, it's easy to use SCREEN to place it out of the way of your graphics windows. You can, however, use it on top of a graphics window - for example, you might want to use EDIT to simulate an edit box in the graphics window. Use gSETWIN to change the **default window** to about the size and position of the desired edit box. The text window moves with it - you must then make it the same size, or slightly smaller, with the SCREEN command. Use 1,1 as the last two arguments to SCREEN, to keep its top left corner fixed. gORDER 1,1 will then bring the default window to the front, and with it the text window. EDIT can then be used.

Here is an example program which uses this technique - moving an 'edit box', hiding it while you edit, then finally letting you move it around.

```
PROC gsetw1:
    LOCAL a$(100),w%,h%,g$(1),factor%,info%(10)
    LOCAL margx%,margy%,chrw%,chrh%,defw%,defh%
    SCREENINFO info%() REM get text window info
    margx%=info%(1) :margy%=info%(2)
    chrw%=info%(7) :chrh%=info%(8)
```

```

defw%=23*chrw%+2*margx%          REM new default window width
defh%=chrh%+2*margy%             REM ... and height
w%=gWIDTH :h%=gHEIGHT
gSETWIN w%/4+margx%,h%/4+margy%,defw%,defh%
SCREEN 23,1,1,1                  REM text window
PRINT "Text win: "; :GET
gCREATE(w%*0.1,h%*0.1,w%*0.8,h%*0.8,1)    REM new window
gPATT -1,gWIDTH,gHEIGHT,0          REM shade it
gAT 2,h%*0.7 :gTMODE 3
gPRINT "Graphics window 2"
gORDER 1,0                      REM back to default+text window
EDIT a$                          REM you can see this edit
gORDER 1,65                      REM to background - could use 9 for Series 3c
CLS
a$=""
PRINT "Hidden: ";
GIPRINT "Edit in hidden edit box"
EDIT a$                          REM YOU CAN'T SEE THIS EDIT
GIPRINT ""
gORDER 1,0 :GET                  REM now here it is
gUSE 1                           REM graphics go to default window
DO                               REM move default/text window around
    CLS
    PRINT "U,D,L,R,Quit";
    g$=UPPER$(GET$)
    IF KMOD=2                     REM Shift key moves quickly
        factor%=10
    ELSE
        factor%=1
    ENDIF
    IF g$="U"
        gSETWIN gORIGINX,gORIGINY-factor%
    ELSEIF g$="D"
        gSETWIN gORIGINX,gORIGINY+factor%
    ELSEIF g$="L"
        gSETWIN gORIGINX-factor%,gORIGINY
    ELSEIF g$="R"
        gSETWIN gORIGINX+factor%,gORIGINY

```

```
ENDIF
UNTIL g$="Q" OR g$=CHR$(27)
ENDP
```